Convolutional Neural Networks Explained from Scratch

Venchislaw

Unemployed bruh

Author Note

All sources used are linked at the end of the paper.

venchislavcodes@gmail.com

Abstract

Convolutional Neural Networks are widely used in practical projects and are found efficient and simple. However most of the people using this architecture do not know how it works under the hood, or have a shallow understanding. This paper is not a research or some new method introduction. It is a simple, beginner-friendly guide to convolutional neural networks.

This paper is written to clarify confusions about Convolutional Neural Networks!

Hope you enjoy it! GitHub - https://github.com/Venchislaw

Why CNN?

UseCase

These networks are commonly used for computer vision problems.

These problems are very specific and differ from structured tabular data ones.

Our images usually vary, so we need to keep our eye on the geometric structure of the data passed, and do it in an efficient way.

Why not MLP?

Ordinary Fully Connected MLP fails both expectations, as data is "flattened" (loss of geometric structure) and the network uses dozens of parameters.

Solution.

To understand CNNs we need to understand how we, human beings see.

One guess says that our brain detects some “significant” features of the object to detect it. Our brain detects edges, corners, line thickness, colors and many other properties to make the final decision. Our brain learned it somehow, learned how to detect these features.

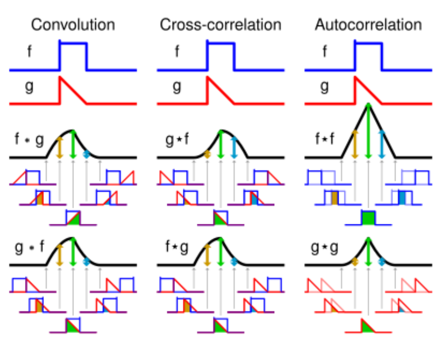
Convolutional Neural Network structure basically aims to learn the model to detect all these properties from the image. It assumes that we apply some “filters” to the image, so We detect edges and other useful information. This Neural Network learns the filters to perform the most efficient feature extraction.

What is CNN?

Convolution

Operation of convolution plays an important role in our network, as the name suggests. There is a math background of this operation to cover, so let’s start with simple things. In math it is an operation on two functions that yields a third function,

such that the output function represents how much the “shape” of the function is changed by the other.



Pic.1 Convolution. Source: WKP[1]

Good intuition on what is a convolution without boring math is a statistics representation. In Stats it is represented as a **weighted moving average.** But what about images? We can apply operation of convolution to the images as long as we assume that our Images are represented with Matrices.

Convolution on the images with various “filters” can give us many interesting outputs. Aha! We’ve already used the word “filter” in this paper. Furthermore, these filters work As a PhotoShop. With the right filter we can blur the image or detect edges or sharpen it Etc. The best thing about it all is that this filter is relatively small and glides across our Image. So, It takes less parameters, keeps our image’s geometric structure and neglects Feature position (yeah, in CNNs position of the object on the image does not matter), so What is the catch? The catch is… It is not called Convolution.

Is it mind blowing? Yes!

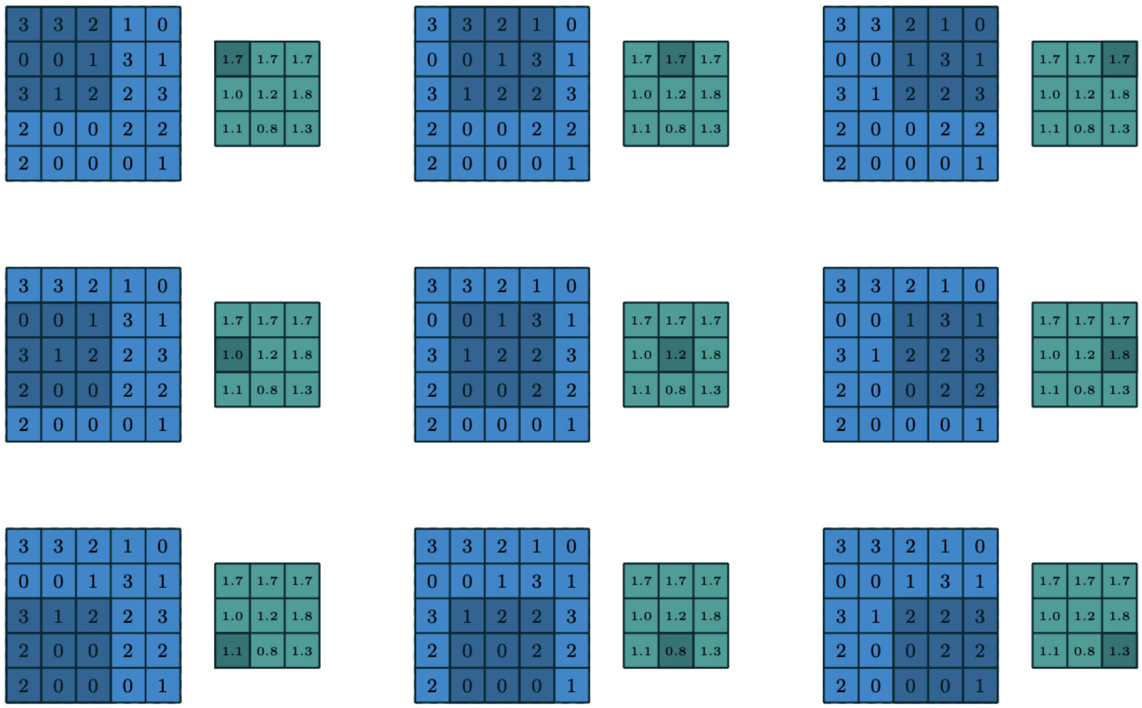
Is it for real? Absolutely!

The trick is - our convolution operation **flips** the second array (filter in this case), but We didn’t do it. It may sound funny and pointless, but in mathematics it plays an Important part, so we can’t neglect it.

But what is the name of this operator if it’s not a convolution?

Cross Correlation! Does absolutely the same thing for images, but without flip.

It is depicted in the first picture.



Pic.2 Convolution on the Image. Source: PWC[2]

Convolution Layer

Let's try to implement a convolutional layer. Let's start with a **grayscale** image.

The most famous example of such grayscale images is the MNIST dataset.

Pic.3 MNIST. Source: WKP[1]

It's just a set of black and white images, each of size 28x28 pixels with pixel values Encoded in range from 0(black) to 255(white). Convolution operation on the layer Is identical to the one on the Pic.2, but we also add a **bias** value.

Convolution operation on the image yields the other image of size:

$$W_o = \frac{W_i + 2p - f}{s} + 1 \quad (1)$$

$$H_o = \frac{H_i + 2p - f}{s} + 1 \quad (2)$$

Let's clarify what these fancy letters are.

W_i - width of the input

W_o - width of the input

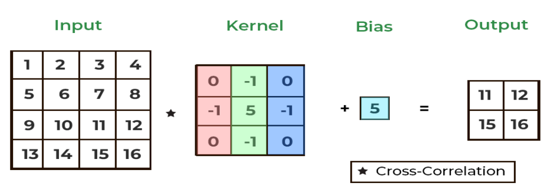
H_i - height of the input

H_o - height of the input

p - padding (we'll clarify it below)

f - filter size (size of our filter)

s - strides (we'll also clarify it)



Pic.3 CNN. Source: GFG[3]

The output in this case is 4x4x1 image. However in Convolutional Neural Nets we can Apply many filters in one layer. In this way we'll produce 4x4xf_n image.

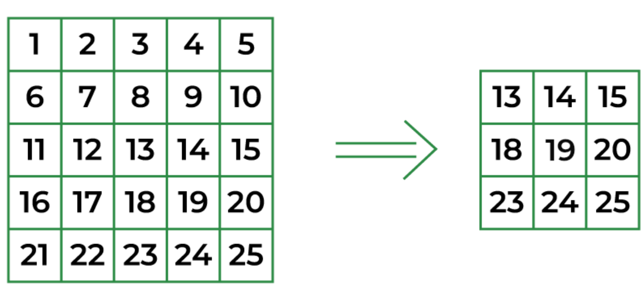
Previously in Notations I mentioned **Padding** and **Strides.** Let's find out what it is!

Padding

If you think deeper about what is happening convolution operation, you'll understand, that we pay less attention to the pixels on the edges, because we glide through them once. That's why we add padding - a "frame" of 0s to process edges more. Also this method Can keep our image size bigger or at least the same. Yep, formulas (1) and (2) suggest, That our image size shrinks, but with the right **padding** value we can keep the image Size the same. We can distinguish three types of padding:

Valid

Means no padding(p=0). Pretty straight forward.



Pic.4 Valid Padding. Source: GFG[3]

Same

Preserves image size from shrinkage. Padding value is calculated as:

$$p = \frac{f-1}{2} \quad (3)$$

If you are interested how did we come up with this equation:

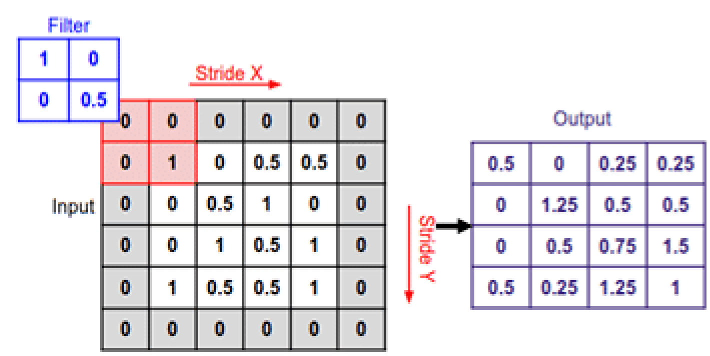
Let's ignore strides (s=1)

$$n + 2p - f + 1 = n$$

$$2p - f + 1 = 0$$

$$2p = f - 1$$

$$p = \frac{f-1}{2}$$



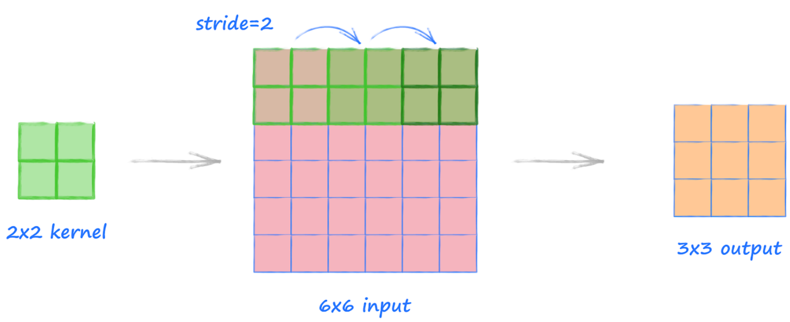
Pic.4 Valid Padding. Source: RSG[4]

Custom

Set padding as a hyperparameter.

Strides

Strides are used to control the output image size. Roughly stride is a step size for our sliding window. If we set s=2, our output image size will be twice smaller (with p=0)



Pic.5 Strides. Source: MYN[5]

Convolution On the colorful Image

Thankfully we see the world in colors. Representation of such images is a 3D Matrix

$$W \times H \times C$$

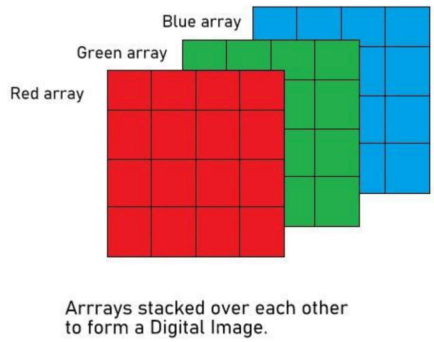
W and H are width and height, respectively, while C is a number of channels.

For colorful images we usually have 3 color Channels: RGB (Red Green Blue) see Pic.6

The solution is really simple. For 3D images we use 3D filters. Each filter chanel slides

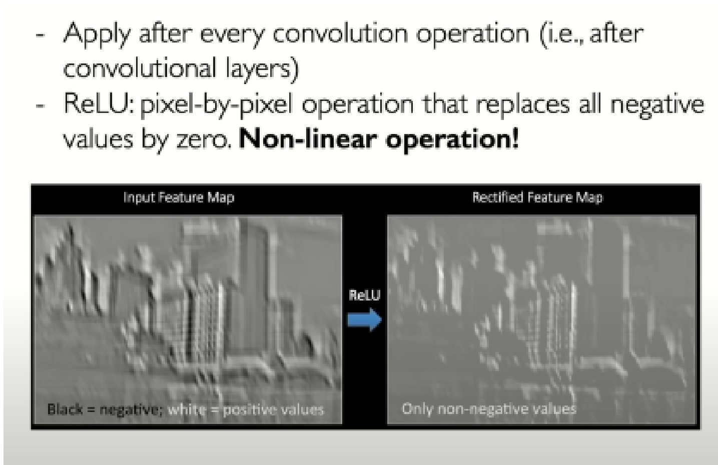
Across the image channel so we produce 2D Matrix.

It may sound like a problem, but it is not. We stack our filters, so it's good for us.



Pic.6 Color Channels. Source: GFG[3]

Finally, as we're working with Neural Networks we apply **activation function** to our Matrices (elementwise). Usually ReLU is used as an activation function.



Pic.7 ReLU Non-Linearity. Source: MIT[6]

Layers In Convolutional Neural Network

Convolutional Neural Networks do not only use Conv Layers. We have few main Types of Layers:

- Convolutional Layer
- Pooling (explained below)
- Dense (fully connected layer from MLP)

References

Lastname, C. (2008). Title of the source without caps except Proper Nouns or: First word after colon. *The Journal or Publication Italicized and Capped*, Vol#(Issue#), Page numbers.

Lastname, O. (2010). Online journal using DOI or digital object identifier. *Main Online Journal Name*, Vol#(Issue#), 159-192. doi: 10.1000/182

Lastname, W. (2009). If there is no DOI use the URL of the main website referenced. *Article Without DOI Reference*, Vol#(Issue#), 166-212. Retrieved from http://www.example.com